An Agent-Based Approach for Supporting Cross-Enterprise Workflows

Liangzhao Zeng, Anne Ngu, Boualem Benatallah

School of Computer Science & Engineering
University of New South Wales
Sydney, NSW 2052, Australia
(zlzhao, anne, boualem)@cse.unsw.edu.au

Milton O'Dell

JustWin Technologies Pty Ltd
Sydney, Australia
modell@justwin.com

Abstract

In order to support global competitiveness and rapid market responsiveness, virtual enterprises need to efficiently integrate different organization's workflows to provide customized services. Currently, most of the integrations are case-based which have high setup cost and involve time consuming low level programming. Cross-enterprise workflow that is able to streamline and coordinate business processes across organizations in dynamic Web environment provides a low cost and flexible solution. We develop an agent-based cross-enterprise workflow Management System (WFMS) architecture which can dynamically integrate the workflows and compose a workflow execution community customized to different workflow specifications.

1. Introduction

The support of cross-enterprise relationships is a key requirement for Business-to-Business (B2B) E-commerce. B2B E-commerce is a prime candidate to take advantage of the information revolution the WWW has brought about. Cheap connectivity and ease of advertising of business services (i.e., processes) on the Web created tremendous opportunities for organizations of any size to diversify their customer-based and become truly global. The electronic availability of business processes (e.g., order procurement, customer relationship management, finance, accounting, human resources, supply chain and manufacturing) has built very high expectations for organizations to increase customer value and establish deeper relationships with partners. There are signs of this phenomenon even today. For example, GM (General Motors) announced in early 2000 that it will move its main operations (i.e, design, manufacturing, selling, and shipping of cars) to the Web [Gen00].

The ability to efficiently and effectively share business processes across the Web is a critical step in the development of the new on-line economy. Organizations must be able to integrate their business processes across boundaries to form what is known as *Virtual Enterprise* [Geo99]. For example, in a supply chain manufacturing environment, the order placement process, order fulfillment process and shipment process might all be undertaken by different companies. It is important for the customer who places an order to have the visibility of the whole process (from ordering to shipping). *Dynamic integration of cross-enterprise business processes* is an essential requirement for organizations to adapt their business practices to the highly volatile and dynamic nature of the Web.

With respect to meeting the requirements of virtual enterprises, we identified two relevant emerging technologies: *workflows* and *agent* technologies. As businesses move more and more of their activities on-line, workflow technology is increasingly gaining relevance as a critical technology for business process reengineering. Workflow Management Systems (WFMSs) have achieved considerable improvements in critical, contemporary measures of performance, such as cost, quality, service, and speed by coordinating and streamlining complex business processes within large organizations (e.g., health-care, education, banking and finance, stock-brokerage, manufacturing, communication, and office automation applications). Current workflow systems are based on the premise that enterprise success requires the management of business processes in their entirety. Indeed, an increasing number of organizations have already automated their internal process management using workflows and enjoyed substantial benefits in doing so. However, one of the most significant weaknesses of existing WFMSs, is the lack of flexible mechanisms to cope adequately with cross-enterprises relationships.

Despite the success of using workflow technology in automating internal business processes, there has been little success to achieve dynamic integration of cross-enterprise

workflows. Indeed, the development of integrated workflows is still ad-hoc, time-consuming and requires enormous effort of low-level programming. This problem is aggravated by the added degree of dynamism, unpredictability, and distribution of the Web. The support of flexible and efficient integration of workflows requires the ability to efficiently discover and exploit business services in a dynamic and constantly growing environment. It also requires the capacity to dynamically establish relationships among business processes.

Agents have emerged recently as an important paradigm for organizing many classes of distributed applications [HS97]. Agents are sophisticated software entities with a high degree of autonomy. They are charged to achieve certain tasks in collaboration with their peers. Agents may be incorporated into an existing application in order to add new functions or customize the execution of existing functions according to specific requirements, operating conditions or observations of user behavior. An agent-based framework will provide component autonomy, multi-party negotiation and pro-activeness. These properties make agent technology one of the most important candidates for providing interoperability and interactions in volatile, dynamic and cooperative environments. However, successful multi-agents system like InfoSleuth[NBN99], Warren/Restina[SKWL99] do not have the concept of a business process model. This means asking a multi-agent system to process workflow requires writing a complete new workflow engine each time which is a daunting task.

In this paper, we propose an approach that combines agents with workflows to effectively *integrate* cross-enterprise workflows. The problem with current workflow technology is the inability to cope with dynamic interactions and interoperability which is what an agent can provide. A cross-enterprise workflow instance can be regarded as a collection of tasks combined in certain ways by a process/control agent. The tasks can be assigned to different service providers and executed in a distributed environment. In our approach, among other things, agents are used to encapsulate (i.e., wrap) services which are able to execute workflow tasks. Based on the requirements of tasks, our system searches for agents with matching capabilities. The relevant agents are used to execute the tasks which are dynamically composed by the system in order to provide the whole service. Agents are not only used to encapsulate services, but also advertise, search and coordinate them. In section 2, we present the model that we use to describe workflow processes. In section 3.1, we present an agent-based workflow architecture. In section 4, we present the execution of a cross-enterprise workflow in the proposed architecture. We discuss related work and summarize our findings in section 5.

2 Workflow Specification Model

We propose a component-based workflow model. Using this model, major parts of a workflow process can be represented as component processes (i.e., tasks or workflows). Thus, a higher-level specification that provides a cross-enterprise workflow process can be composed from existing component workflows. Workflow components can be reused and specialized in different organization settings.
A *workflow* process is represented as a tuple W:
$W = (T, E_w, R_w)$ where:

1. *T is a set of tasks:* $T = \{t_1, t_2, ...t_n\}$
2. *E_w is a set of workflow Event-Condition-Action(ECA) rules:* $E_w = \{e_1, e_2, ...e_n\}$. The workflow ECA rule specifies the coordination among the tasks. In order to define the ECA rules, we use the following operations:
   - Δt: Enables the execution of the task t;
   - ∇t: Disables the execution of the task t;
   - $\rightsquigarrow t$: Sends a message to the task t;
   - $\star W$: Starts the execution of the workflow W;
   - $\odot W$: Finishes the execution of the workflow W.

   A workflow ECA's rule is defined as follows:

   $$(t_i.result == R) \Longrightarrow \Delta t_j,$$

   which means enable the execution of task t_j when the execution the task t_i returns R.
3. R_w is the result of the workflow execution. It can be `success`, `failure` or `null`. It is initialized to `null` before a workflow begins.

A *task* is represented as a tuple t_i:
$t_i = (N_i, P_i, O_i, E_{ti}, D_i, R_{ti})$ where:

1. N_i is the name of the task, which is a mandatory string to identify the task.
2. P_i is type of the task, which can be `required` or `optional`. All the `required` tasks must be executed and completed successfully. A workflow can still proceed even if an `optional` task was failed or not been executed.
3. O_i is the task object, which is a tuple (P, S, TR, OP) where:
   (a) P is a set of properties that represent general information about the task object (e.g., the creator of the task).
   (b) S is a set of possible states: $S = \{s_1, s_2, ...s_n\}$, where:

i. s_i is the initial state: $s_i \in S$

ii. s_e is the final state: $s_e \in S$, if object's state changes to s_e, it implies that the execution of the task is successful.

iii. s_f is the failure state: $s_f \in S$, if object's state changes to s_f, it implies that the task fails to be executed.

(c) TR is set of possible transitions (s_i, s_j), where $s_i, s_j \in S$.

(d) OP is the set of operations on S, such that $TR \subseteq S \times OP \times S$

4. *E_{ti} is a set of ECA rules:* $E_{ti} = \{e_{i1}, e_{i2}, ...e_{in}\}$. The ECA rules are used to define how the task coordinates or synchronizes with other tasks. Two kinds of rules are supplied:

   - $C \longrightarrow A$: if condition C is true, then operation A *might* be executed.
   - $C \Longrightarrow A$: if condition C is true, then operation A *must* be executed.

5. D_i is the deadline of the task.

6. R_{ti} is the result of the task execution (i.e, `success`, `failure`, or `null`) which is set initially to `null`.

In this model, we distinguish workflow ECA rules from task ECA rules. Workflow ECA rules are used to specify the interaction among workflow tasks, for example when to synchronize, when to make a choice. Task ECA rules are used to guide the execution procedure of individual tasks, for example when to trigger related operations. It should be noted that a task can be atomic or complex (i.e., a workflow). In our model, the definition of a cross-enterprise workflow involves the identification of the tasks that compose the workflow and specification of the interactions between them. This is different from traditional workflow where it is also necessary to define who will be responsible for the execution of each task and how much time is allocated for each task at specification time. In our approach, tasks are dynamically assigned to service providers (i.e, entities that can perform the tasks, e.g., programs) during the enactment of a cross-enterprise workflow. The dynamic composition of tasks to provide a cross-enterprise workflow will be discussed further in section 4.

3 Agent-based Workflow Architecture

3.1 The Architecture

The architecture we propose to support cross-enterprise workflows is shown in Figure 1. There are three logical components in this architecture: workflow definition tool, agent society and actual service. **Workflow Definition Tool** is responsible for defining cross-enterprise workflow specifications. It allows a user to chart a personalized cross-enterprise workflow using a Web Browser based on existing templates. If none of the existing template is suitable, a new template need to be defined by a power user. Once the workflow specification is verified to be syntactically correct, it is translated into the format as specified in Section 2.

The **Agent Society** is a collection of agents that provide the general functionalities of a cross-enterprise workflow execution engine. There are three types of agents in the agent society, namely *process agent, discovery agent and monitor agent.* **Actual Services** offer applications (services) that allow users to access and perform tasks (e.g., order procurement, customer relationship management, finance, accounting, human resources, supply chain and manufacturing). In our work, we are interested in services that are accessible through the Web. In order to abstract proprietary services from their physical organizations, we wrap them in service agents. The use of agents to execute cross-enterprise workflows is presented in the next section. Due to space constraints, the description of discovery agents is outside the scope of this paper.

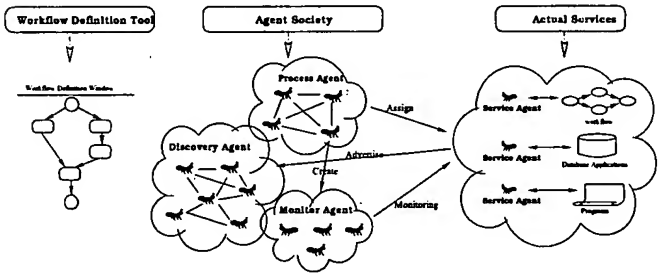

Figure 1. Agent-based WFMS Architecture

3.2 Service Agent

A service agent encapsulates a task that can be executed by a service provider. It contains the following information: (i) service identification, which indicates the port that the service agent listens to for requests, the format of the request message that it can understand and process(i.e its query processing capabilities), (ii) agent capabilities which specify the underlying service, and (iii) agent properties which specify service's constraints. For example, the flight ticket booking service can be represented in a service agent as shown in the figure 2. A service agent only accepts queries involving objects and operations that it advertised. It translates such a query into one or more requests that the actual service provider can understand. Upon receiving an-

Agent identity
agent name: Flight_agent
agent address: 203.4.5.101:2323
agent type: service
agent interface: XML, SQL92
Agent capabilities
supported object: flight_ticket
supported operations: reserve, unreserve, book, unbook
supported query: price
Agent properties
agent constraint 1: departure ∈ {Sydney,Melbourne}
agent constraint 2: destination ∈ {Hongkong, Beijing, NewYork}
... ...

Figure 2. Service Agent

swers for those local requests, it processes the answers if necessary, maps them to results in terms of what the requesting agent can understand.

3.3 Process Agent

The process agent is in charge of transforming a workflow specification into a workflow instance. As such, it has a number of duties and responsibilities:

- The process agent assigns tasks to the relevant service agents. This is done by making a query to the discovery agent and finding the relevant available service agents. When a process agent receives many choices for performing a particular task, it lets the user who initiated that particular task to make the final decision. It then makes connections to the service agent and sends task specifications to it.
- Once a task is assigned to a service agent, a monitor agent is then created and sent to the task execution site. After the monitor agents migrate to the task execution sites, the process agent will instantiate the task ECA rules.
- The process agent manages the execution flow of the tasks according to the workflow's ECA rules. It coordinates the tasks to achieve a certain goal. It can enable, disable, suspend or resume the tasks according to the workflow ECA rules.
- During the execution of the workflow, process agent receives all the event messages from the monitor agents. This will enable user to be aware of the status of the workflow instance without having to subscribe to any service agent.
- The process agent can forward the workflow specifications to another process agent when it does not have the ability to process the specification. For example, a process agent specializes in design workflow will not be able to process a supply-chain workflow.

3.4 Monitor Agent

A monitor agent performs the actual monitoring of the execution of a given task at a local site. It has the following functionalities:

- It migrates to the site where the corresponding task is executed.
- It downloads the ECA rules of the corresponding task from the process agent. These ECA rules will guide the service agent in executing the task.
- The monitor agent can be called back and updated by the process agent. And it also can re-migrate to continue its monitoring function at the local site.
- It informs the process agent and other monitor agents about the execution status of a task. This is done by sending messages to them when the service agent changes status of the task that it is responsible for execution.

4 Execution of Cross-Enterprise workflows

During the life cycle of a workflow instance, the above four type of agents will form a community according to the specification of a particular workflow, work load of individual agent and the quality of services those agents can provide. The community disbands after the termination of the workflow execution.

Figure 3. Business Trip Planing Workflow

4.1 An Example

Consider a business trip workflow application which is used by a salesman as shown in Figure 3. It involves interaction among agents from an airline company, a hotel company, and a car rental company. The business workflow has the following constraints:

- The salesman can book the flight ticket and hotel at the same time.

- If flight booking or hotel booking is not successful, the trip cannot proceed.
- If both bookings are successful, there is an optional operation: Car Booking. The workflow can still proceed even if the car can not be rented

Using the workflow model defined in section 2, this trip planning workflow W can be declaratively defined as follows:

Workflow $W = \{T, E_w, R_w\}$, where:
$T = \{t_1, t_2, t_3\}$
$E_w = \{e_{w1}, e_{w2}\}$, where
$e_{w1} : *W \Longrightarrow \Delta t_1 \wedge t_2$
$e_{w2} : ((t_1.result == success) \wedge (t_2.result == success)) \Longrightarrow \Delta t_3$
$R_w = null$

The tasks :
$t_1 = \{N_1, P_1, O_1, E_{t1}, D_1, R_{t1}\}$, where
N_1 is flight ticket booking, P_1 is required, O_1 is Flight Ticket, defined as:

States	:	{request,reserved,unavailable,booked}
Operations		
reserve	:	(request,reserved)
unreserve	:	(reserved,request)
book	:	(reserved,booked)
unbook	:	(booked, request)

$E_{t1} = \{e_{11}, e_{12}, e_{13}\}$,where
$e_{11} : (t_1.Ticket.state == reserved) \Longrightarrow (\rightsquigarrow t_2)$,
$e_{12} : (t_2.Room.state == reserved) \longrightarrow (t_1.Ticket.book)$,
$e_{13} : (t_1.result == failure) \Longrightarrow$
$(t_2.Room.unreserve \vee t_2.Room.unbook)$.
D_1=05 May 2000, $R_{t1} = null$.

$t_2 = (N_2, P_2, O_2, E_{t2}, D_2, R_{t2})$, where
N_2 is hotel booking, P_2 is required,
O_2 is Hotel Room, defined as:

States	:	{request,reserved, unavailable,booked }
Operations		
reserve	:	(request, reserved)
unreserve	:	(reserved,request)
book	:	(reserved, booked)
unbook	:	(booked, request)

$E_{t2} = \{e_{21}, e_{22}, e_{23}\}$,where
$e_{21} : (t_2.Room.state == reserved) \Longrightarrow (\rightsquigarrow t_1)$
$e_{22} : (t_1.Ticket.state == reserved) \longrightarrow (t_2.Room.book)$
$e_{23} : (t_2.result == failure) \Longrightarrow$
$(t_1.Ticket.unreserve \vee t_1.Ticket.unbook)$.
D_2=10 May 2000, $R_{t2} = null$.

$t_3 = (N_3, P_3, O_3, E_{t3}, D_3, R_{t3})$, where
N_3 is car booking, P_3 is optional,
O_3 is Car, defined as:

States	:	{request, unavailable, booked }
Operations		
book	:	(request,booked)
unbook	:	(booked, request)

$E_{t3} = \phi$, D_3=10 May 2000, $R_{t3} = null$.

4.2 Composition Procedure

We assume that the workflow integrator is used to define the business trip workflow specification, namely, W. An instance of the workflow is executed as follows:

Parsing Workflow Specifications

The process agent receives the definition W from the user, then parses it. The workflow W consists of three tasks: t_1, t_2 are required tasks, t_3 is an optional task. The main responsibility of the process agent is to assign the given tasks to the available service agents on the Web.

Searching For Service Agents

The process agent queries the discovery agent for a suitable service agent which can execute the task t_1. The content of the query message that it sends to the discovery agent is shown in Figure 4.

Message Type	:	query
Agent Identity	:	?
Group Identity	:	?
Task Object	:	$t_1.Ticket$
Search	:	all
Hop	:	3

Figure 4. Query Message

Upon receiving this message, the discovery agent searches its yellowpage and catalogue repositories (since the query indicates searching through all repositories of a discovery agent), then returns the result to the process agent as shown in Figure 5. The hop count is set to 3, which means that the request will be propagated to at least three discovery agents. The above message indicates that the process agent discovers one service agent and one group service agents which can execute the task.

Message Type	:	reply
Yellowpage	:	(ip=203.5.1.2,port=1334, agent name=Flight_agent)
Catalogue	:	(ip=230.0.0.1,port=3333, group name=FlightTicket)

Figure 5. Result Message

Assigning Tasks

Assigning a task to a service agent has two phases.

1. Negotiation Phase
   In this phase, the process agent sends a proposal message as shown in Figure 6 to the service agents(Flight_agent, Traveller) and the service agent group(FlightTicket). The service agents who are able to execute the task will send the corresponding response messages as shown in Figure 7.
2. Task Assignation Phase
   In this phase, the process agent presents the responses to the user and let the user select the most appropriate

Message Type	:	proposal
Sender	:	203.5.15.21:1234:Process_agent
Task Object	:	$t_1.Ticket$
Price	:	?

Figure 6. Pr p sal Message

Message Type	:	response
Sender	:	203.5.1.2:1334:Flight_agent
Task	:	$t_1.Ticket$
Price	:	A$1000

Figure 7. Response Message

service agent. It then assigns the task to the service agent (which is selected to execute the task) by sending the assignment message shown in Figure 8. In this message, the process agent assigns the task t_1 to the service agent Flight_agent. Here, task t_1 is a tuple as we defined in section 4.1 which has all the detailed information about the task. After the service

Message Type	:	assign
Sender	:	203.5.15.21:1234:Process_agent
Task	:	t_1
Receiver	:	203.5.1.2:1334:Flight_agent

Figure 8. Assign Message

agent receives the `assign` message, it will send a confirmation message to the process agent.

Creating Monitor Agents

For every task that is assigned to the service agent, the process agent creates a monitor agent for it. The monitor agent migrates to the site of the service agent (which is selected to execute the task). After the monitor agent arrives at the service agent site, it will send a confirmation message to the process agent to signal that it is ready to monitor the task.

Instantiating the ECA rules

The task ECA rules do not have any execution context information and can not be executed when users first define these rules. Only after monitor agents have migrated to the tasks' execution site, the rules can be instantiated by the process agent. For example, assuming the tasks t_1, t_2, t_3 have been assigned to service agents `Flight_agent`, `Hotel_agent`, `Car_agent`, and monitor agents MA_1, MA_2, MA_3 have migrated to tasks' execution sites respectively, then the ECA rule e_{11}:

$(t_2.Room.state == reserved) \longrightarrow (t_1.Ticket.book)$

can be instantiated to

$(t_2(MA_2)Room.state == reserved) \longrightarrow (t_1(MA_1)Ticket.book);$

This instantiated task's ECA rules will be downloaded by the monitor agents. After all the tasks have been assigned to the service agents, all the monitor agents have been created and migrated to the tasks' execution sites and have downloaded the instantiated task ECA rules, the workflow agent-community is ready to execute the cross-enterprise workflow instance. Note that the composed agent-community is not a static entity. It might dynamically change during the execution of the workflow. For example, if the assigned service agent fails to execute the task, the process agent can locate an alternative service agent and continue executing the workflow.

4.3 Distributed Enactment of a Cross-Enterprise Workflow

Normally, workflow engine acts as a central task coordinator. Based on the specification, it creates process instance and task list, controls over the execution flow of the tasks and coordinates task execution as well. Such a central control model obviously has some drawbacks when task execution is carried out in a widely distributed network environment, especially when exceptions arise during the coordination and the task execution. These drawbacks include: central server is a single point of failure; when exception occurs, central server needs to suspend the whole workflow instance to handle it.

We propose a distributed coordination approach which adopts the mobile agent technology. We separate the task control flow and the task coordination: the process agent implements the task execution flow control; the monitor agents act as task coordinators to be distributed in all the task execution sites.

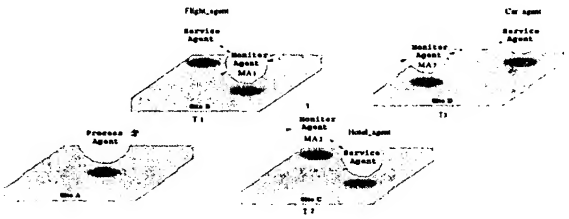

Figure 9. Distributed Coordination

In the following, we use a simple example to explain how distributed coordination works. Here, we assume workflow agent-community has been formed for the workflow specification W, four tasks have been assigned to three service agents (`Flight_agent`, `Hotel_agent`, `Car_agent`) respectively, three monitor agents (MA_1, MA_2, MA_3) have been created and migrated to each of the task execution sites (Figure 9), all the tasks' ECA rules have been instantiated and downloaded

by the monitor agents.

Starting the Workflow

At the beginning, the process agent will execute the first workflow ECA rule:

$$e_{w1} : \star W \Longrightarrow \Delta t_1(MA_1) \wedge \Delta t_2(MA_2)$$

The action $\Delta t_1(MA_1)$ and $\Delta t_2(MA_2)$ results in sending messages to monitor agent MA_1 and MA_2 which will inform service agent `Flight_agent` and `Hotel_agent` to start executing their tasks.

Executing the Task t_1 and t_2

`Flight_agent` and `Hotel_agent` begin to execute their tasks after they received the enable signal (Δt_i) message from the process agent. We assume that `Flight_agent` reserves the ticket first. The Object Ticket's state will be changed to `reserved`, it will then trigger the action $\rightsquigarrow t_2(MA_2)$ which will send event message (Figure 10) to the monitor agent MA_2. This event satisfies the condition of $t_3.e_{32}$ which is a '$\longrightarrow$' type of rule and does not trigger an object's changing states event but indicates that the object's state can be changed from `reserved` to `booked`. So, if the service agent `Hotel_agent` has already reserved the room, it can issue the booking operation now.

Message Type	:	event
Sender	:	203.5.1.2:2234:MA_1
Event	:	t_1(Flight_agent).Ticket.state = reserved
Receiver	:	203.5.25.21:2234:MA_2

Figure 10. Event Message

If both `Flight_agent` and `Hotel_agent` make the booking, the results from both task executions are successful. The monitor agents MA_1, MA_2 located at `Flight_agent` and `Hotel_agent` will relay messages to process agent about the their tasks' state and trigger the dependent task which happens to be task t_3. On the other hand, the service agent may not be able to make the required bookings, for example, no ticket might be available from service agent `Flight_agent`. In this situation, if the deadline constraint is not violated, the process agent will seek alternative service agent to perform task t_1 and re-instantiate t_1's ECA rules. The monitor agent MA_1 will update its ECA rules and migrate to the new task execution site. If task t_1 is overdue, the task t_1's result will be a failure. MA_1 will be informed of such a failure by MA_2 and it will execute the task's ECA rule e_{22}:

$(t_1(Flight\_agent).result == failure) \Longrightarrow$
$(t_2(Hotel\_agent).Room.unreserve \vee$
$t_2(Hotel\_agent).Room.unbook)$

This is a '$\Longrightarrow$' type of ECA rule which means it will trigger the indicated events and force the service agent `Hotel_agent` to issue operation either `unreserve` or `unbook`.

Executing the Task t_3

Task t_3 is an optional task. If the service agent `Car_agent` successfully executes the task, the whole workflow will finish. If `Car_agent` can not book the car, process agent can either find another service agent or choose to skip that task.

Finishing the Workflow

After task t_3 gets the result, process agent will finish the workflow and pass the result of execution to the user.

5 Related Work and Conclusions

Techniques related to integrating business processes exist in several fields including EDI, component-based E-commerce systems, databases, agents, and groupware systems.

EDI aims at offering an automatic and standard way to transfer data between business partners. EDI investigated static solutions that are appropriate if the business systems to be integrated belong to organizations that have *long-term* and *static* trading relationships. Component-based E-commerce systems [Dog98] typically rely on distributed object middleware technologies such as CORBA and DCOM. They focus on the integration of a small number of tightly coupled applications. However, they present several limitations that make them ineffective when the service space is large and highly dynamic (e.g., the cost to set up a new business relationship is very high, it is not presently possible to dynamically integrate new business services, etc.)

Other advanced approaches to dynamic composition of cross-organization workflows include the CMI[GSCB99] project at MCC and eFlow[CIJ+00] at HP labs. In CMI, they proposed a Service Oriented Process model and a set of service management primitives such as `activity place holder`, `dynamic role assignment`, `repeated option creator`, `awareness events`, `process synchrony` etc. The goal is to be able to provide a framework for flexible, plug and play approach to cross-organization workflow composition. However, CMI has not addressed the issue of brokering and selection of services that goes beyond what is stated in the service interfaces.

eFlow is a platform that supports the specification, enactment, and management of composite e-services. A composite service is described as a process schema that composes other basic or composite services. It aims to provide a flexible, configurable and open approach to cross-organization

workflow composition. The flexibility is achieved mainly by allowing dynamic service process modification and binding of services through dynamic service discovery. Both CMI and eFlow emphasis on a process model that allows incomplete or abstract specification of the workflow at design time to allow for more dynamic adaptation at runtime.

The use of agents in executing a single organization WFMS have been discussed in several publications, e.g. in [CS96, JFJ+96, PM99]. In [CS96], each workflow is represented by multiple personal agents, actor agents and authorization agents. These agents act as personal assistants performing actions on behalf of the workflow participants and facilitating interaction with other participants or organization specific WFMS. In [JFJ+96], the multi-agent architecture consists of a number of autonomous agencies. A single agency consists of a set of subsidiary agencies which is controlled by one responsible agent. Each agent is able to perform one or more services. These atomic agents can be combined to form complex services by adding ordering constraints and conditional control. However, neither [CS96] nor [JFJ+96] adopts agent technology to compose workflow execution engine dynamically. The logic for workflow processing is hard-coded and thus it is hard to reuse this workflow execution engine for other business processes. In [PM99], a multi-plan state machine agent model is presented. Agents are assembled dynamically from the descriptions in a blueprint language and can be modified while running. But users have to describe in detail how agents can be assembled together and what changes are allowed in priori.

In our approach, the agent-community for a specific workflow execution is optimally and automatically composed based on the context of workflow execution and can self-adapt and react to changes during the execution. Our approach aims to achieve the similar result as CMI and eFlow, with the added benefit of scalability, distribution and interoperability as inherent in multi-agents system. We show how the workflow agent-community gets constructed through an example. We illustrate how the agent-community executes the workflow specification and modifies themselves during the execution of the workflows. We also show a novel distributed monitoring of cross-enterprise workflow using monitor agents. This enables user to be aware of the status of the workflow instance without having to pay the high subscription cost to any service agent.

We are currently implementing prototype for composing workflows based on XML and EJB (Enterprise Java Beans) technologies.

References

[CIJ+00] Fabio Casati, Ski Ilnicki, Li-Jie Jin, Vasudev Krishnamoorthy, and Ming-Chien Shan. eFlow: a Platform for Developing and Managing Composite e-Services. Technical Report HPL-2000-36, HP Laboratoris Palo Alto, 2000.

[CS96] J. Chang and C. Scott. Agent-based workflow: TRP support Environment (TSE). *Computer Networks and ISDN Systems*, 28(7-11):1501–1511, 1996.

[Dog98] A. Dogac, editor. *ACM SGMOD Record: Special Issue on Electronic Commerce*, ACM SIGMOD RECORD. ACM, December 1998. 27(4).

[Gen00] General Mortors web page, 2000. http://www.gm.com.

[Geo99] D. Georgakopoulos, editor. *Information Technology for Virtual Enterprises*, Proc. of the 9th Int. Workshop on Research Issues on Data Engineering. IEEE Computer Society, March 1999.

[GSCB99] D. Georgakopoulos, H. Schuster, A. Cichocki, and D. Baker. Managing process and service fusion in virtual enterprises. *Information System, Special Issue on Information System Support for Electronic Commerce*, 24(6):429–456, 1999.

[HS97] M. Huhns and M. Singh, editors. *Internet-Based Agents*, IEEE Internet Computing. IEEE, July 1997. Special Issue on Agents.

[JFJ+96] N. R. Jennings, P. Faratin, M. J. Johnson, T. J. Norman, P. O'Brien, and M. E. Wiegand. Agent-based business process management. *International Journal of Cooperative Information Systems*, 5(2&3):105–130, 1996.

[NBN99] Marian Nodine, Bill Bohrer, and Anne Hee Hiong Ngu. Semantic brokering over dynamic heterogeneous data sources in infosleuth. In *Proceedings of the 15th International Conference on Data Engineering*, March 1999.

[PM99] Krzysztof Palacz and Dan C. Marinescu. An agent-based workflow management system. Technical report, Computer Sciences Department, Purdue University, 1999.

[SKWL99] Katia Sycara, Matthias Klush, Seth Widoff, and Jianguo Lu. Dynamic service matchmaking among agents in open information environments. *SIGMOD Record*, 28(1), March 1999.